INFO8002

Large-Scale Data Systems

Exercise Session #2

Academic year 2021-2022

LIÈGE université

Reminder

REMINDER :

Distributed System Models

Distributed system models are defined as the combination of **(i)** a process abstraction, **(ii)** a link abstraction, and **(iii)** a failure detector abstraction:

- **Fail-stop**
  - Process : **Crash-stop**
  - Link : **Perfect**
  - Failure Detector : **Perfect**
- **Fail-silent**
  - Process : **Crash-stop**
  - Link : **Perfect**
  - Failure Detector : **/**
- **Fail-noisy**
  - Process : **Crash-stop**
  - Link : **Perfect links**
  - Failure Detector : **Eventually perfect**
- **Fail-recovery**
  - Process : **Crash-recovery**
  - Link : **Stubborn**
  - Failure Detector : **/**
- **Fail-arbitrary**
  - Process : **Byzantine**
  - Link : **Perfect**
  - Failure Detector : **/**

REMINDER :

Shared Memory Abstraction

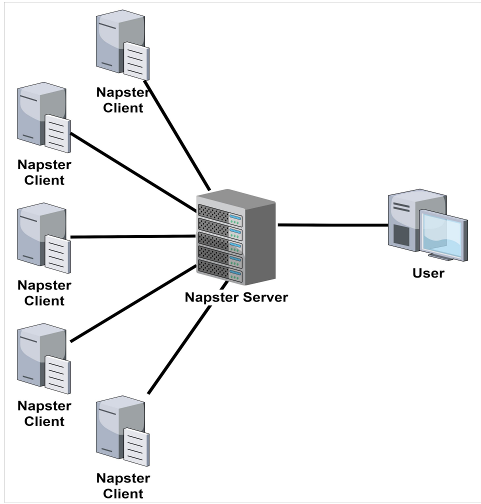
- **Shared memory** can be viewed as **an array of registers** to which a process can read or write.
- **Shared memory models** are defined as a combination of behaviour of registers in presence of **(i)** failures, and **(ii)** concurrent operations :

  - **Safe Register (not seen)**
    - Failures : **??**
    - Concurrency : **Arbitrary** value
  - **Regular Register**
    - Failures : **Fail-stop or Fail-silent (others?)**
    - Concurrency: **Previous** or **concurrently** written value
  - **Atomic Register**
    - Failures : **Fail-stop (others?)**
    - Concurrency : Ensure **linearisability** of operations

PROBLEM 1

Content Sharing System

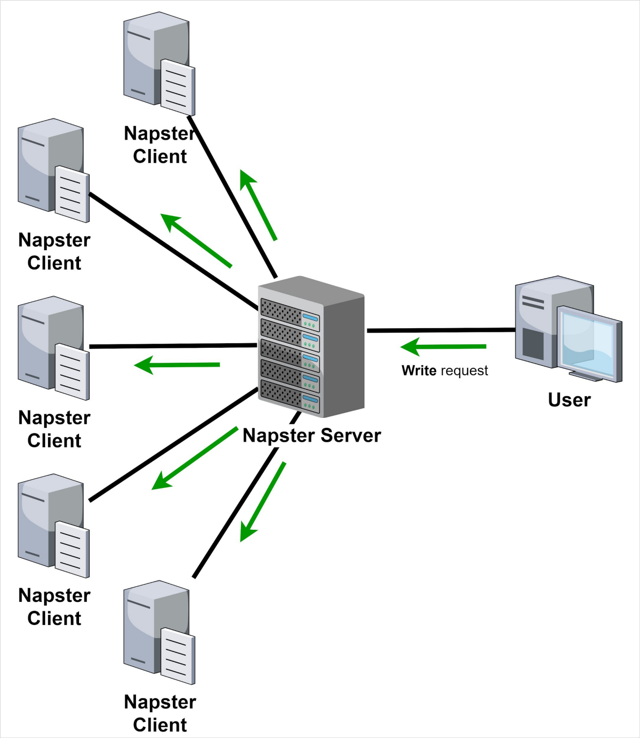
You are responsible for designing a system allowing the **storage** and **distribution** of content.



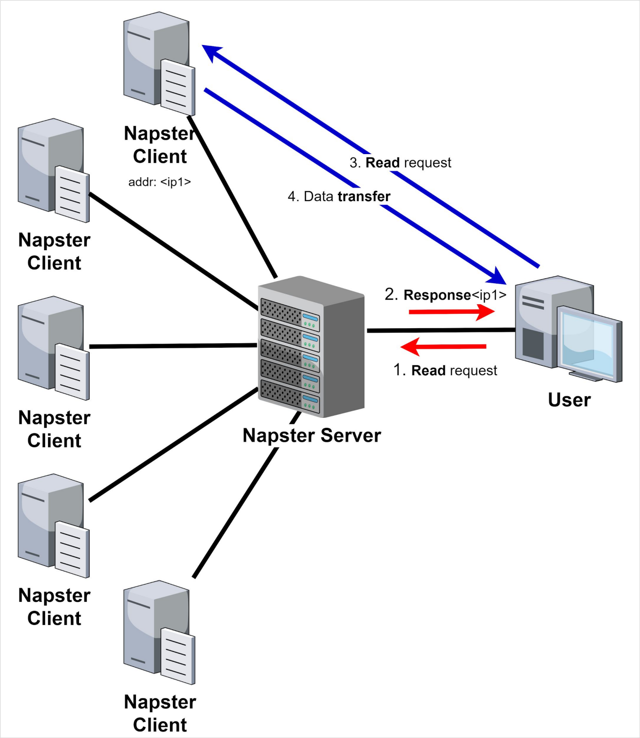
Specify an architecture for this distributed storage system and **provide** a pseudo-implementation using shared registers.

Content Sharing System

Write query

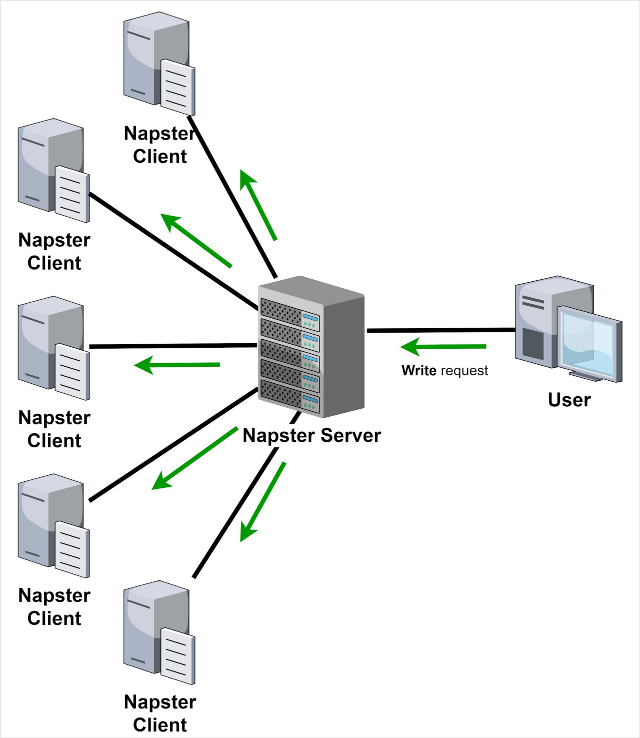


Read query

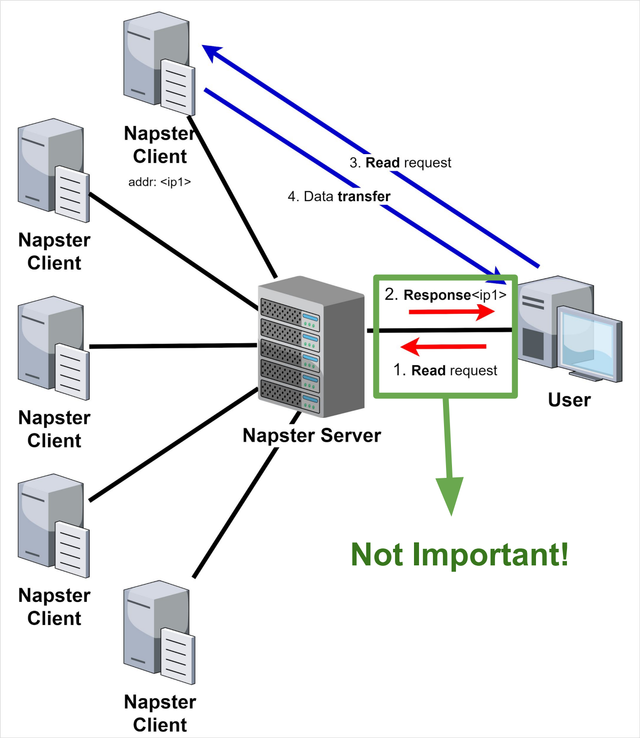


Content Sharing System

Write query



Read query



Content Sharing System

Problems:

1. **How can we ensure that operations terminates ?**

Content Sharing System

Problems:

1. **How can we ensure that operations terminates ?**

   Property n°1: **"Termination"**

   "If a correct process invokes an operation, then the operation eventually completes."

2. **How can we ensure that users receives a coherent response to their read request ?**

Content Sharing System

Problems:

1. **How can we ensure that operations terminates ?**

   Property n°1: **"Termination"**

   "If a correct process invokes an operation, then the operation eventually completes."

2. **How can we ensure that users receives a coherent response to their read request ?**

   Property n°2: **"Validity"**

   *"A **read** that is not concurrent with a **write** returns the last value written; a **read** that is concurrent with a **write** returns the last value written or the value concurrently written."*

Content Sharing System

Module Specification:

Module 1: Interface and properties of distributed storage

Module:

Name: *NapsterClientServer*, **instance** *np*.

Events:

Request: < np, Read | r, m > **:** Invokes a **read** operation on **m** consecutive registers starting on register **r**.

Request: < np, Write | v, r> **:** Invokes a **write** operation with value **v** starting on register **r**.

Indication: < np, ReadReturn | v > **:** Completes a **read** operation with return value **v**.

Indication: < np, WriteReturn> **:** Completes a **write** operation.

Properties:

NP1: Termination.

NP2: Validity.

} *"(1, N) Regular Register"* Module

Content Sharing System

Implementation

Algorithm 1:

Implements:

NapsterClientServer, **instance** *np*.

Uses:

(1, N)-RegularRegister, **instance** *onrr*.

upon event < *np*, *Init* **> do**

???

upon event < *np*, Write | v, r > **do**

???

upon event < *np*, Read | r, m > **do**

???

upon event < *onrr*, ReadReturn | r, v> **do**

???

upon event < *onrr*, WriteReturn | r > **do**

???

Content Sharing System

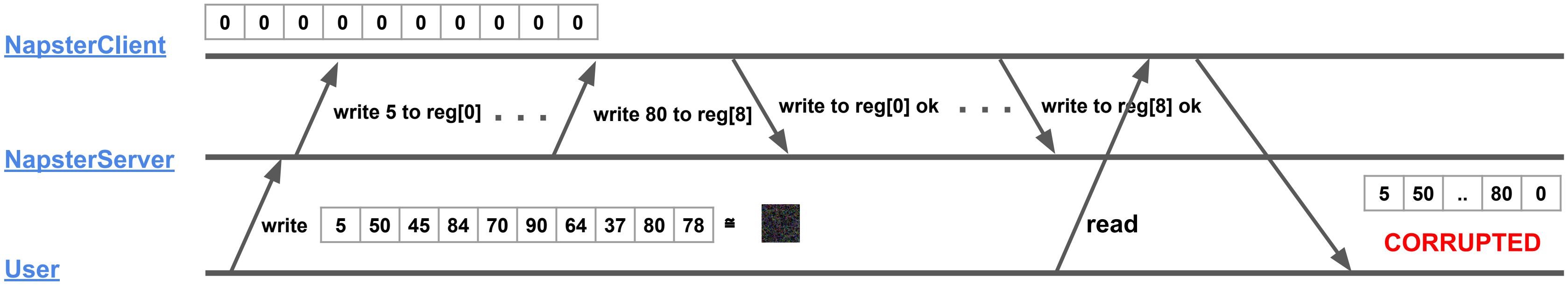
Implementation

Algorithm 1:

```
Implements:
    NapsterClientServer, instance np.
Uses:
    (1, N)-RegularRegister, instance onrr.

upon event < np, Init > do
    memory := [0]^MemorySize;
    pendingR := ∅;
    pendingWr := ∅;

upon event < np, Write | v, r > do
    forall v' ∈ v do
        pendingWr := pendingWr ∪ {r + index(v)};
    forall v' ∈ v do
        trigger < onrr, Write | v', r + index(v)>;

upon event < np, Read | r, m > do
    ReadRet := [0]^m;
    offset := r;
    for i in range(m) do  pendingR := pendingR ∪ {r + i};
    for i in range(m) do  trigger < onrr, Read | r + i >;

upon event < onrr, ReadReturn | r, v> do
    pendingR := pendingR \ {r};
    ReadRet[r-offset] := v;
    if pendingR ⊆ ∅  then
        trigger < np, ReadReturn | ReadRet >;

upon event < onrr, WriteReturn | r > do
    pendingWr := pendingWr \ {r};
    if pendingWr ⊆ ∅  then
        trigger < np, WriteReturn >;
```

Content Sharing System

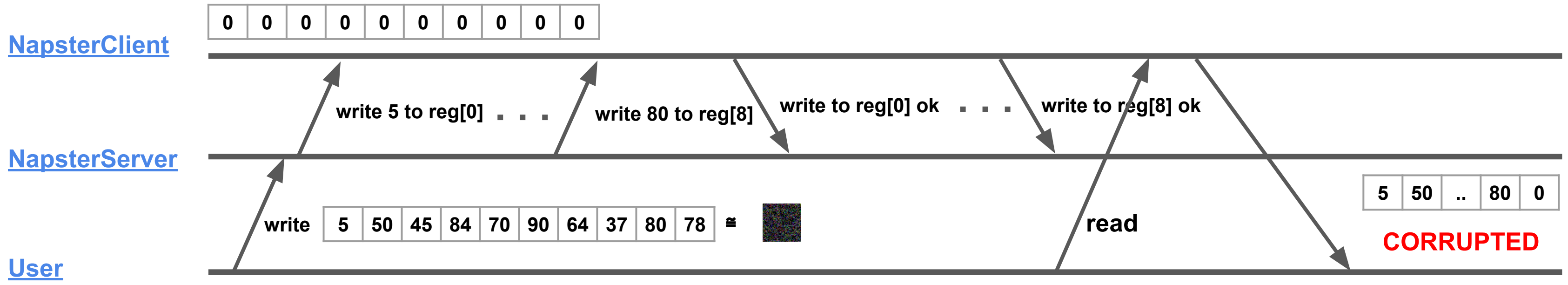
PROBLEM:



⟶ We would like to ensure **coherency** of the distributed storage.

Content Sharing System

PROBLEM:



⟶ We would like to ensure **coherency** of the distributed storage.

SOLUTION: Write to **temporary buffer** until all data are received.

Content Sharing System

Implementation

Algorithm 2:

Implements:

(1, N)-NewRegularRegister, **instance** *onnrr*.

Uses:

BestEffortBroadcast, **instance** *beb*;

PerfectPointToPointLinks, **instance** *pl*;

PerfectFailureDetector, **instance** *P*.

upon event < *onnrr*, *Init* **> do**

val **:=** **[0]**$^{\text{MemorySize}}$;

wrBuffer := **[0]**$^{\text{MemorySize}}$;

correct := Π;

writeset := ∅;

upon event < *onnrr*, Write | v, r **> do**

trigger < *beb*, Broadcast | [WRITE, v, r]>;

upon event < *beb*, Deliver | q, [Write, v, r]> **do**

wrBuffer[r] = v;

trigger < *pl*, Send | q, ACK>;

upon event < *onnrr*, Flush **> do**

val := wrBuffer;

trigger < *onnrr*, FlushReturn >;

… (cfr. Theoretical Lectures : "*(1,N)-RegularRegister*")

Content Sharing System

Implementation

Algorithm 3:

```
Implements:
    NapsterClientServer, instance np.
Uses:
    (1, N)-NewRegularRegister, instance onnrr.

upon event < np, Init > do
    pendingR := pendingWr := ∅;

upon event < np, Write | v, r > do
    forall v’ ∈ v do
        pendingWr := pendingWr ∪ {r + index(v)};
    forall v’ ∈ v do
        trigger < onnrr, Write | v’, r + index(v)>;

upon event < np, Read | r, m > do
    ReadRet := [0]^m; offset := r;
    for i in range(m) do  pendingR := pendingR ∪ {r + i};
    for i in range(m) do  trigger < onrr, Read | r + i >;

upon event < onrr, ReadReturn | r, v> do
    pendingR := pendingR \ {r};
    ReadRet[r-offset] := v;
    if pendingR ⊆ ∅  then
        trigger < np, ReadReturn | ReadRet >;

upon event < onrr, WriteReturn | r > do
    pendingWr := pendingWr \ {r};
    if pendingWr ⊆ ∅  then
            trigger < onnrr, Flush>

upon event < onnrr, FlushReturn > do
        trigger < np, WriteReturn >;
```

Content Sharing System

Problems:

- **Do you see any other problems?**

Content Sharing System

Problems:

- **Napster** is a technology allowing to have several servers that can be reached for write **and** read requests. How would you implement this in:
    1. In a **fail-stop** system?
    2. In a **fail-silent** system?
    3. In a **byzantine** system?

HOMEWORK !